Systême soft

Программный комплекс Систэм Платформ

SePlatform.Data Server 2.1
Модуль логики

Руководство администратора

Редакция	Соответствует версии ПО
3. Предварительная	2.1.2

Содержание

1. О модуле

Основное назначение модуля - логико-вычислительная обработка собранных оперативных данных. Возможности модуля определяются возможностями языка SePlatform.Om.

Настройка модуля

Модуль имеет общие и дополнительные параметры.

1. Общие	
Имя модуля	**LogicsModule**
Идентификатор модуля	**LogicsModule**
Активность	**Да**
Уровень трассировки в журнал приложений	**Информационные сообщения**
Вести журнал работы модуля	**Нет**
Размер журнала работы модуля, МБ	**10**
Количество дополнительных журналов работы	**1**
2. Дополнительные	
Режим работы	**Сигнальная модель**
Неявное приведение целочисленного значения к булевому	**Нет**
Регистрозависимость исходного кода	**Да**
Игнорировать правила приведения типов для деклараций	**Нет**
Разрешить неявные преобразования из примитивных типов в вариант	**Нет**
Режим вывода вещественных типов	**Неявно**
Стратегия приведения целочисленных типов при использовании арифметических операторов	**Все уровни**

Общие параметры

Параметр	Описание
Имя модуля	Название модуля
Идентификатор модуля	Идентификатор модуля
Активность	Активность модуля: › **«Да»** - модуль запущен › **«Нет»** - модуль остановлен
Уровень трассировки в журнал приложений	Типы сообщений, которые фиксируются в журнал приложений: › **«Предупреждения и аварийные сообщения»** - логические ошибки, ошибки работы модуля. Предупреждения содержат некритичные ошибки. Аварийные сообщения информируют об ошибках, которые влияют на работоспособность службы › **«Информационные сообщения»** - сообщения, которые показывают основную информацию о работе модуля › **«Отладочные сообщения»** - сообщения, которые наиболее детально отражают информацию о работе модуля Вышестоящий уровень входит в состав нижестоящего: если выбрано **«Информационные сообщения»**, то в журнал фиксируются **«Предупреждения и аварийные сообщения»** и **«Информационные сообщения»**

Параметр	Описание
Вести журнал работы модуля	Ведётся ли журнал работы модуля: › **«Да»** › **«Нет»**
Размер журнала работы модуля, МБ	Ограничение на размер файла журнала работы модуля в мегабайтах. При достижении максимального размера создается новый файл, копия старого файла хранится на рабочем диске
Количество дополнительных журналов работы	Количество файлов заполненных журналов работы модуля. Минимальное значение - 1, максимальное - 255

Дополнительные параметры

Параметр	Описание
Режим работы	Выбирается в зависимости от способа построения проекта автоматизации: › **«Сигнальная модель»** - выбирается, когда объекты автоматизации и их параметры выражаются в виде сигналов, для формирования дерева сигналов используется SePlatform.Data Server › **«Объектная модель»** - выбирается в случае применения объектно-ориентированного подхода к построению проекта с помощью SePlatform.Development Studio
Неявное приведение целочисленного значения к булевому	Флаг обратной совместимости: › **«Да»** - целочисленные значения могут быть неявно приведены к булевым (режим обратной совместимости) › **«Нет»** - целочисленные значения не могут быть неявно приведены к булевым Значение по умолчанию - **«Нет»**.
Регистрозависимость исходного кода	Флаг обратной совместимости: › **«Да»** - регистр символов имеет значение › **«Нет»** - регистр символов не имеет значения (режим обратной совместимости) Значение по умолчанию - **«Да»**. После таблицы приведены примечания к работе флагов обратной совместимости (стр. 7).

Параметр	Описание
Игнорировать правила приведения типов для деклараций	Флаг обратной совместимости: › **«Да»** - если при объявлении переменной ей присваивается значение, то правила приведения типов игнорируются (режим обратной совместимости). › **«Нет»** - при объявлении переменной с присваиванием ей значения действуют правила приведения типов. Значение по умолчанию - **«Нет»**. После таблицы приведены примечания к работе флагов обратной совместимости (стр. 7).
Разрешить неявные преобразования из примитивных типов в вариант	Флаг обратной совместимости: › **«Да»** - преобразования разрешены (режим обратной совместимости). › **«Нет»** - преобразования запрещены. Значение по умолчанию - **«Нет»**. После таблицы приведены примечания к работе флагов обратной совместимости (стр. 7).
Режим вывода вещественных типов	Стратегия выбора типа для вещественных значений: › **«Явно»** - если у значения нет суффикса, то значение будет иметь тип double, если есть суффикс f или F, то тип float. Например, в выражении x = y * 0.5f / 0.3 значение 0.5f будет иметь тип float, а 0.3 - тип double. › **«Неявно»** - тип вещественного значения компилятор выбирает автоматически, суффиксы не поддерживаются. Значение по умолчанию - **«Неявно»**.

Параметр	Описание
Стратегия приведения целочисленных типов при использовании арифметических операторов	Правило определения типа результата арифметической операции над целыми числами: › **«Все уровни»** - результатом арифметической операции может быть целое число размером: › 1 байт (Int1/Uint1); › 2 байта (Int2/Uint2); › 4 байта (Int4/Uint4); › 8 байт (Int8/Uint8). Тип результата принимается равным типу аргумента бóльшей мощности, а размер значения результата не учитывается. › **«Два последних уровня»** - результатом арифметической операции может быть целое число размером: › 4 байта (Int4/Uint4); › 8 байт (Int8/Uint8). Тип результата принимается равным типу аргумента бóльшей мощности, но не менее 4 байт. Если аргумент бóльшей мощности имеет тип меньше 4 байт, то тип результата принимается равным 4 байта. Если размер значения результата арифметической операции превышает 4 байта, то тип результата принимается равным 8 байт. ОБРАТИТЕ ВНИМАНИЕ Значение по умолчанию - **«Два последних уровня»**. Значение **«Все уровни»** следует устанавливать в случае, если результат арифметической операции присваивается переменной типа Int1/Uint1 или Int2/Uint2.

Примечания к работе флагов обратной совместимости

ОБРАТИТЕ ВНИМАНИЕ

Если **Регистрозависимость исходного кода** отключена (режим обратной совместимости):

› Нельзя обратиться к пространству имён Variant.

› Для обращения к пространству имён String используйте имя Str.

ОБРАТИТЕ ВНИМАНИЕ

Не рекомендуется включать флаг **Игнорировать правила приведения типов для деклараций**, так как это может привести к потере точности данных.

ОБРАТИТЕ ВНИМАНИЕ

Если в формуле есть операция, в которой операнды имеют недопустимый тип, и включён флаг **Разрешить неявные преобразования из примитивных типов в вариант**, то операнды будут приведены к типу variant, а результат выполнения операции будет VT_EMPTY (неопределённое значение).

Например, выражение 100 && 200 будет успешно скомпилировано.

Если флаг выключен, произойдёт ошибка компиляции (правильное поведение).

ОБРАТИТЕ ВНИМАНИЕ

Если выполняется логическая операция над целыми числами и включены флаги **Неявное приведение целочисленного значения к булевому** и **Разрешить неявные преобразования из примитивных типов в вариант**, то в операции операнды будут приведены к логическому типу (а не к типу variant) и операция вернёт логическое значение true или false.

Например, в выражении 100 && 200 значения 100 и 200 будут преобразованы в true и результатом вычисления значения выражения будет true.

2. Задачи модуля

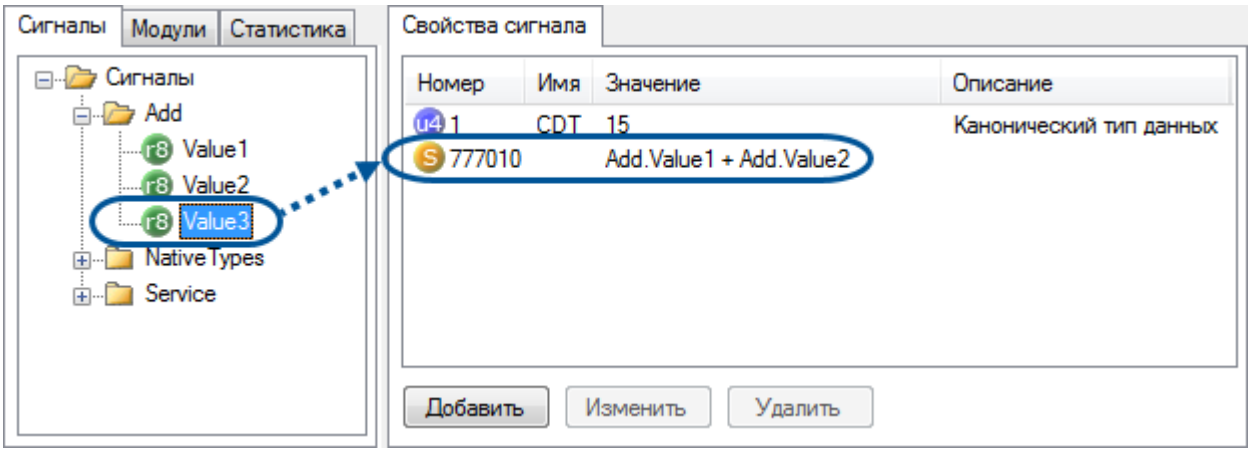
Модуль выполняет свои задачи после настройки специальных сигнальных свойств.

2.1. Вычисление значения сигнала по формуле

Чтобы значение сигнала вычислялось по формуле добавьте сигналу свойство:

- 777010 (тип String) - формула для вычисления значения сигнала.

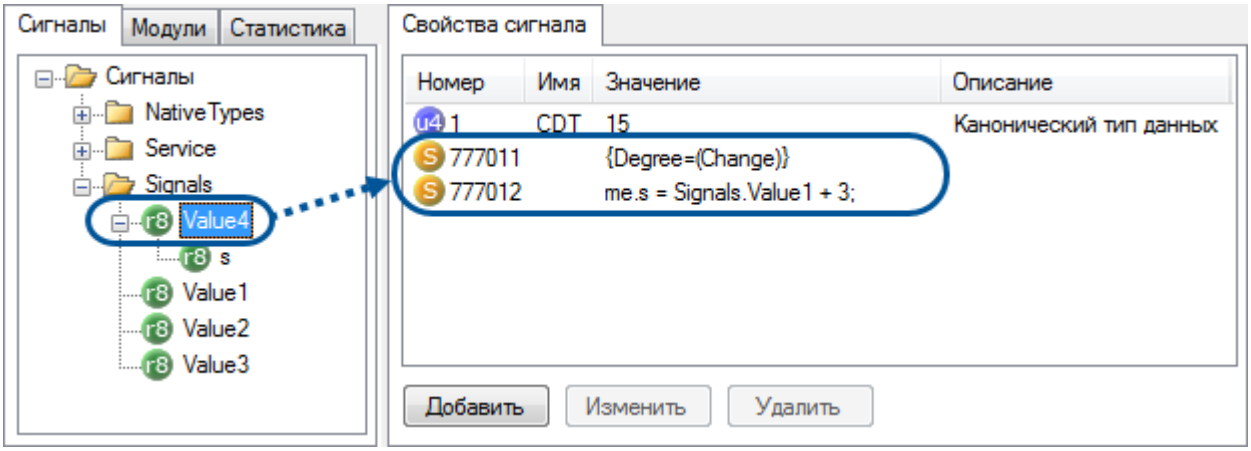
Значение сигнала будет пересчитываться при изменении значения любого сигнала, используемого в формуле.



2.2. Выполнение процедуры и условия её активации

Чтобы процедура вычислялась по заданному условию, добавьте сигналу следующие свойства:

- 777012 (тип String) - код процедуры;
- 777011 (тип String) - условие выполнения. Возможные значения:
  - **«{Degree=(Change)}»** - процедура выполняется только если изменилось значение свойства Value либо Quality данного сигнала
  - **«{Degree=(AnyChange)}»** - процедура выполняется даже если изменилась только метка времени данного сигнала
  - **«{Degree=(Update)}»** - процедура выполняется даже если ни одно из свойств сигнала не изменилось.

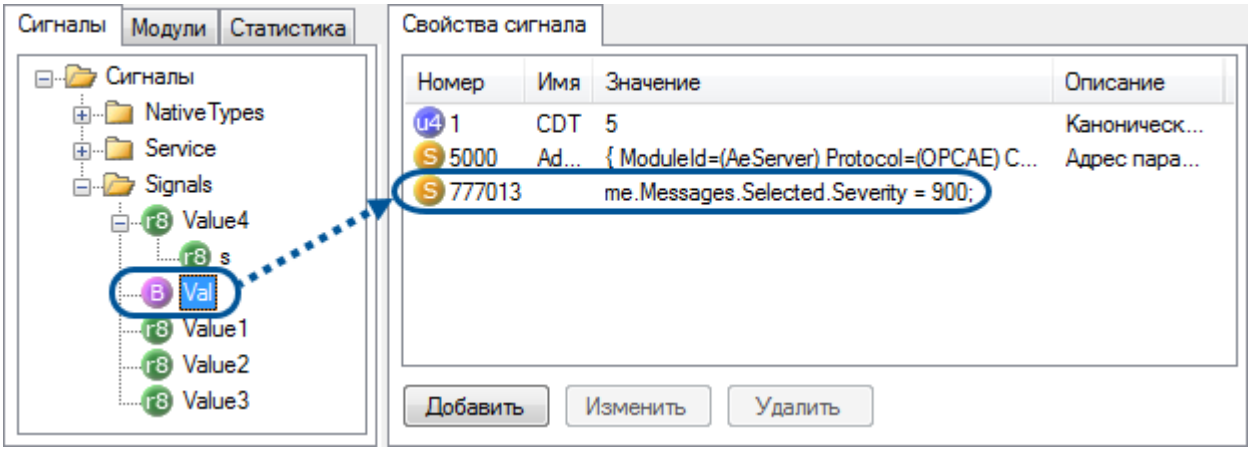


2.3. Выполнение процедуры перед отправкой события

Чтобы процедура выполнялась перед отправкой события (в тот момент, когда по сигналу уже сгенерировано событие, но само событие ещё не отправлено клиенту), запишите в сигнальное свойство 777013 исполняемый код процедуры. В момент запуска процедуры для сигнала создается и заполняется структура me.Messages.Selected, которая представляет из себя массив атрибутов сгенерированного события.

Атрибуты структуры me.Messages.Selected:

Атрибут	**Описание**
Message	Строка, содержащая текст события
Severity	Целое беззнаковое 4 байтовое число, содержащее значение важности
Condition	Строка, содержащая имя условия события
Subcondition	Строка, содержащая имя подусловия события



2.4. Включение и отключение генерации событий

Чтобы включить/отключить генерацию событий по некоторым технологическим объектам (или отдельным условиям технологического объекта), используйте методы Enable/Disable структуры Messages:

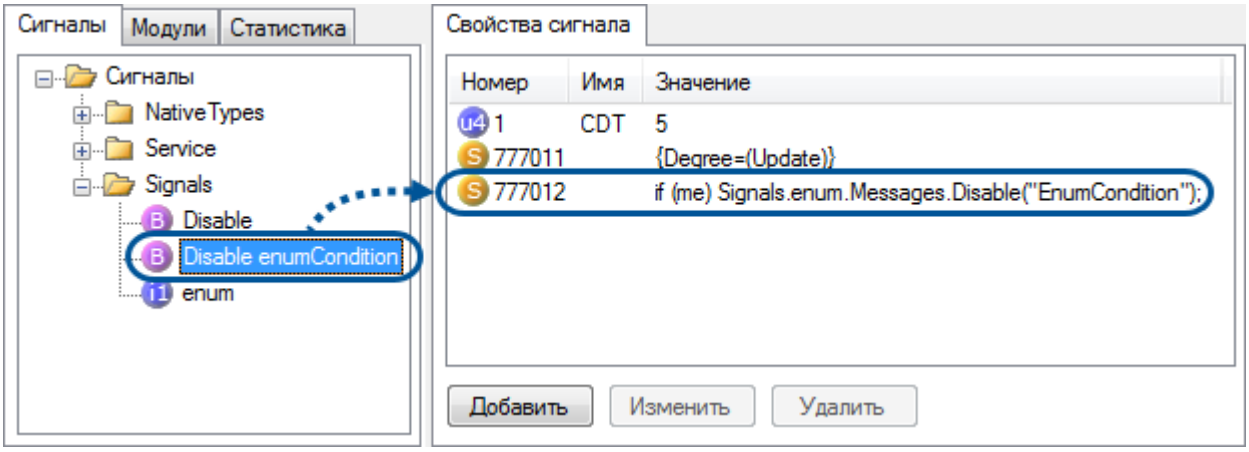
Метод	Описание
Enable()	Включить генерацию событий по всем условиям объекта
Disable()	Отключить генерацию событий по всем условиям объекта
Enable (ConditionName)	Включить генерацию событий по условию ConditionName (ConditionName - строка с именем условия)
Disable (ConditionName)	Отключить генерацию событий по условию ConditionName (ConditionName - строка с именем условия)

ПРИМЕР

Отключить генерацию событий по условию EnumCondition:

1. Создайте сигнал типа bool.
2. Добавьте сигналу свойства 777011 и 777012.
3. В свойстве 777011 запишите условие активации процедуры.
4. В свойстве 777012 запишите код процедуры:

```
«if(me) Signals.enum.Messages.Disable("EnumCondition");»
```

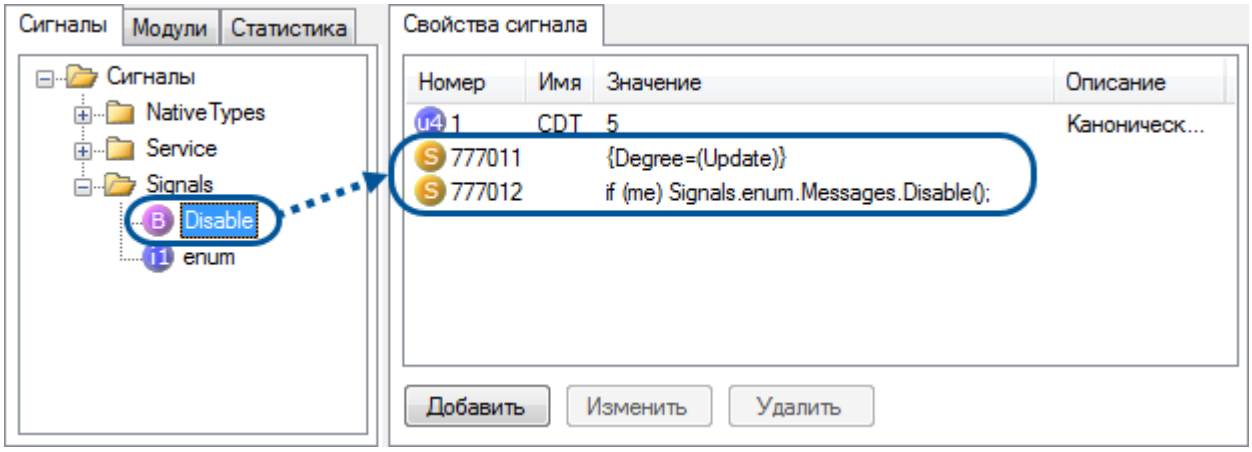


ПРИМЕР

Отключить генерацию событий по всем условиям:

1. Создайте сигнал типа bool.
2. Добавьте сигналу свойства 777011 и 777012.
3. В свойстве 777011 запишите условие активации процедуры.
4. В свойстве 777012 запишите код процедуры:

```
«if (me) Signals.enum.Messages.Disable();»
```



2.5. Выполнение процедуры по таймеру

Чтобы процедура выполнялась циклически по таймеру, добавьте сигналу следующие свойства:

- 777016 (тип String) - код исполняемой процедуры
- 777015 (тип UInt4) - период срабатывания таймера в миллисекундах

Минимальный период - 100мс, при указании меньшего значения используется минимальное допустимое значение.

ОБРАТИТЕ ВНИМАНИЕ

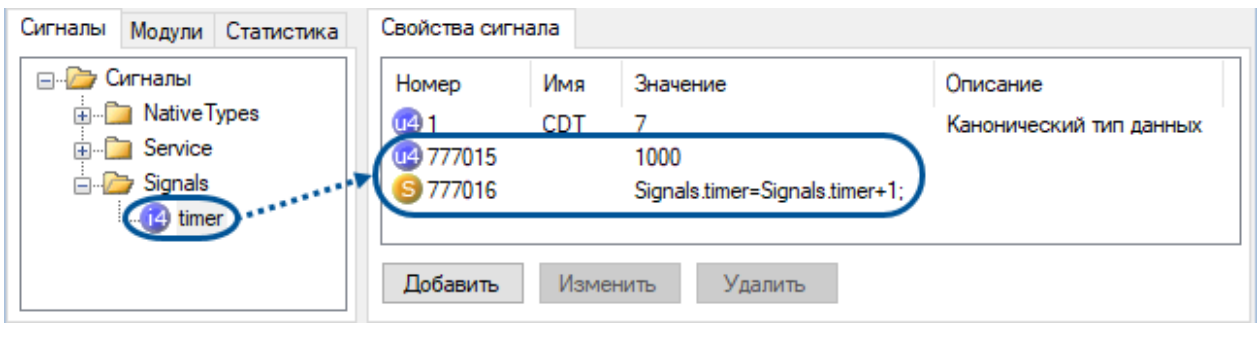
Не рекомендуется использовать много таймеров с периодом менее 1000мс.

Если период не задан (свойство 777015 отсутствует), процедура выполняется с периодом 1000мс.

Таймер и связанная с ним процедура начинают работать сразу после запуска SePlatform.Data Server.

ПРИМЕР

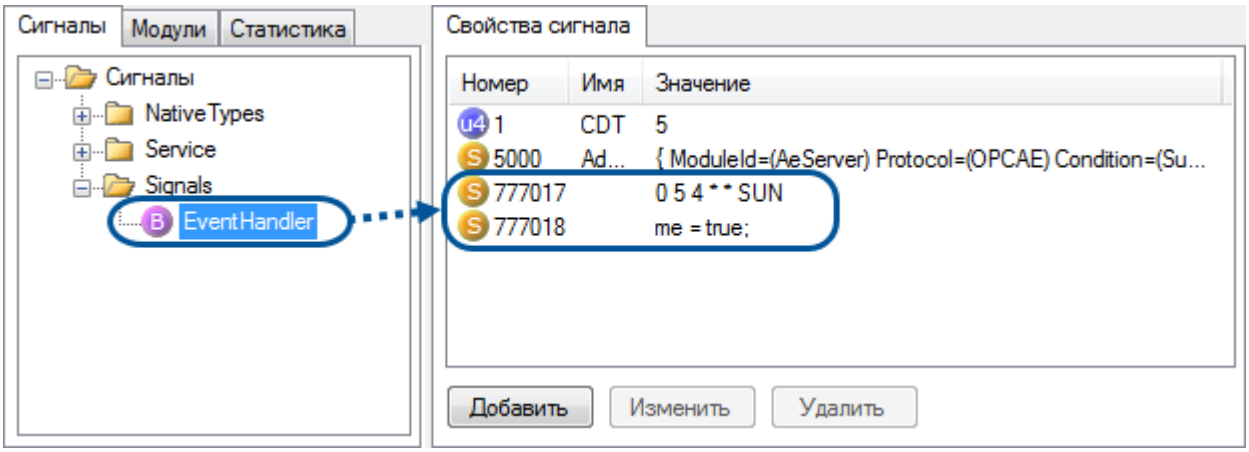
На рисунке ниже показана процедура, которая ежесекундно увеличивает значение сигнала на единицу.



2.6. Выполнение процедуры по расписанию

Чтобы процедура выполнялась по расписанию, запишите исполняемый код процедуры в сигнальное свойство 777018, а cron-выражение укажите в сигнальном свойстве 777017.

На рисунке ниже показан пример cron-выражения для выполнения процедуры каждое воскресенье в 04:05.



ПРИМЕР

Примеры cron-выражений:

- Каждый день в 10:15:05:

```
5 15 10 * * ?
```

- С 1-го по 20-е число каждого месяца в 2:00, в 8:00, в 9:00 и в 10:00:

```
0 0 2,8-10 1-20 * ?
```

- В 1:00 каждый день с понедельника по пятницу включительно (2 варианта):

```
0 0 1 ? * 1-5
```

```
0 0 1 ? * MON-FRI
```

2.7. Вызов функций из внешних динамических библиотек

Для вызова внешних функции из DLL-библиотек, используются сигнальные свойства:

- 777005 - для определения внешних функций
- 777006 - для составления карты дескрипторов сигнатур внешних функций
- 777012 - для вызова функции в теле процедуры

2.7.1. Определение внешних функций

Внешние функции определяются в сигнальном свойстве 777005. Ниже показан пример определения двух внешних функций:

```
{ Id=(0) Name=(FirstExternalFunction) DllFunctionName=(GetComputerNameW) LibraryName=
(kernel32.dll) }
{ Id=(1) Name=(test_ExtLibCalc) DllFunctionName=(ExtLibCalc) LibraryName=(extlib.dll) }
```

Поле	Описание
Id	Идентификатор дескриптора внешней функции
Name	Имя функции для последующего вызова из скрипта
DllFunctionName	Имя функции в DLL-файле
LibraryName	Имя DLL-файла, из которого вызывается внешняя функция

2.7.2. Составление карты дескрипторов

Карта дескрипторов составляется в сигнальном свойстве 777006. Ниже показана структура записи карты дескрипторов сигнатур двух внешних функций в формате JSON:

```
{
  "0": {
    <Определение дескриптора внешней функции с идентификатором 0>
  }
  "1": {
    <Определение дескриптора внешней функции с идентификатором 1>
  }
}
```

Чтобы определить дескрипторы в сигнальном свойстве 777006 используйте параметры:

Параметр	Описание
CallingConvention	Способ вызова функции: › **«0»** - Default (CDecl); › **«1»** - WinApi (StdCall).
Charset	Кодировка символов: › **«0»** - UTF-8. Однобайтные кодовые позиции › **«1»** - UTF-16. Двухбайтные кодовые позиции › **«2»** - ANSI. Однобайтные кодовые позиции. Кодировка - согласно настроенной в системе кодовой странице.
ParameterDirection	Направление параметра: › **«1»** - входной параметр (In); › **«2»** - выходной параметр (Out); › **«3»** - входной и выходной параметр (Inout).

Параметр	Описание
TypeCode	Код типа данных: › **«0»** - void; › **«1»** - int1; › **«2»** - uint1; › **«4»** - string; › **«5»** - bool; › **«5»** - SizedBool1; › **«9»** - int2; › **«10»** - uint2; › **«13»** - SizedBool2; › **«17»** - int4; › **«18»** - uint4; › **«19»** - float4; › **«21»** - SizedBool4; › **«25»** - int8; › **«26»** - uint8; › **«27»** - float8; › **«29»** - SizedBool8; › **«32»** - TimeStamp.
StringMarshalKind	Тип маршалинга строки: › **«0»** - отсутствует; › **«1»** - с применением буфера. Параметр можно не указывать, если TypeCode не равен 4 (string).

2.7.3. Вызов функций

Внешние функции, определенные в сигнальных свойствах 777005 и 777006, вызываются в ходе выполнения процедур, код которых определен в сигнальном свойстве 777012:

```
rez: int4 = me.FirstExternalFunction(&rezstr, &siz);
```

2.7.4. Пример 1

Вызов внешней функции WinAPI GetComputerNameW из библиотеки `kernel32.dll`.
Сигнатура функции:

```
BOOL GetComputerNameW(
  _Out_   LPTSTR  lpBuffer,
  _Inout_ LPDWORD lpnSize
);
```

1. Определите внешнюю функцию (в свойстве 777005):

```
{ Id=(3) Name=(FirstExternalFunction) DllFunctionName=(GetComputerNameW) LibraryName=
(kernel32.dll) }
```

2. Определите дескриптор для вызываемой внешней функции (в свойстве 777006):

```
{
  "3":{
    "CallingConvention":"1", //способ вызова функции: WinAPI
    "Charset":"1", //кодировка UTF-16 (заявлена поставщиком DLL)
    "Return":{ //возвращаемое значение
      "TypeDescriptor":{
        "TypeCode":"5", //тип bool
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0" //маршалинг не применяется
          }
        }
      }
    },
    "Parameters":[
      { //параметры функции
        "TypeDescriptor":{
          "TypeCode":"4", //тип string (соответствует LPTSTR)
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"1"//маршалинг с буфером
            }
          }
        },
        "ParameterDirection":"2" //направление параметра - Out
      },
      {
        "TypeDescriptor":{
          "TypeCode":"18", //тип uint4 (соответствует LPDWORD)
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"0" //маршалинг не применяется
            }
          }
        },
        "ParameterDirection":"3" //направление параметра - Inout
      }
    ]
  }
}
```

3. Вызовите внешнюю функцию в коде процедуры (в сигнальном свойстве 777012):

```
rezstr :string = Str.Reserve(255); //создать пустую строку с резервом памяти
siz: uint4 = 255;
rez: int4 = me.FirstExternalFunction(&rezstr, &siz); //вызвать внешнюю функцию
me.out = rezstr; //записать значение буфера в сигнал
```

2.7.5. Пример 2

Вызов внешней функции WinAPI ExtLibCalc из библиотеки `extlib.dll`.

Сигнатура функции:

```
std::int32_t __stdcall ExtLibCalc(
  wchar_t const *inputString,
  double a1,
  double x1,
  double a2,
  double x2,
  double *y1,
  double *y2,
  wchar_t *error,
  std::uint32_t errorBufferSize
)
```

1. Разместите библиотеку `extlib.dll` в папке `<папка SePlatform.Data Server>\Server`.
2. Определите внешнюю функцию ExtLibCalc в сигнальном свойстве 777005:

```
{ Id=(1) Name=(test_ExtLibCalc) DllFunctionName=(ExtLibCalc) LibraryName=(extlib.dll) }
```

3. Определите дескриптор для вызываемой внешней функции (в свойстве 777006):

```
{
  "1":{
    "CallingConvention":"1", //способ вызова функции: WinAPI
    "Charset":"1", //кодировка UTF-16 (заявлена поставщиком DLL)
    "Return":{ //возвращаемое значение
      "TypeDescriptor":{
        "TypeCode":"17", ////тип int4
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0"
          }
        }
      }
    },
    "Parameters":[
      { //параметры функции
        "TypeDescriptor":{
          "TypeCode":"4", //тип string
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"0"
            }
          }
        },
        "ParameterDirection":"1" // входной параметр (In)
      },
      {
        "TypeDescriptor":{
          "TypeCode":"27", //тип float8
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"0"
            }
          }
        },
        "ParameterDirection":"1" // входной параметр (In)
      },
      {
        "TypeDescriptor":{
          "TypeCode":"27", //тип float8
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"0"
            }
          }
        },
        "ParameterDirection":"1" // входной параметр (In)
      },
      {
        "TypeDescriptor":{
```

```
        "TypeCode":"27", //тип float8
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0"
          }
        }
      },
      "ParameterDirection":"1" // входной параметр (In)
    },
    {
      "TypeDescriptor":{
        "TypeCode":"27",
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0"
          }
        }
      },
      "ParameterDirection":"1" //входной параметр (In)
    },
    {
      "TypeDescriptor":{
        "TypeCode":"27", //тип float8
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0"
          }
        }
      },
      "ParameterDirection":"2" // выходной параметр (Out)
    },
    {
      "TypeDescriptor":{
        "TypeCode":"27", //тип float8
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"0"
          }
        }
      },
      "ParameterDirection":"2" // выходной параметр (Out)
    },
    {
      "TypeDescriptor":{
        "TypeCode":"4", //тип string
        "TypeMarshalOptions":{
          "StringMarshalOptions":{
            "StringMarshalKind":"1"
          }
        }
      },
```

```
        "ParameterDirection":"3" // входной и выходной параметр (Inout)
      },
      {
        "TypeDescriptor":{
          "TypeCode":"18", //тип uint4
          "TypeMarshalOptions":{
            "StringMarshalOptions":{
              "StringMarshalKind":"0"
            }
          }
        },
        "ParameterDirection":"1" // входной параметр (In)
      }
    ]
  }
}
```

3.1. Вызовите внешнюю функцию в коде процедуры (в свойстве 777012):

```
// Подготовка параметров для вызов функции
_str: string = me.inputString;
_a1: double = me.a1;
_x1: double = me.x1;
_a2: double = me.a2;
_x2: double = me.x2;
_y1: double = 0;
_y2: double = 0;
_err: string = Str.Reserve(100);
_length :uint4 = Str.Length(_err);
// Вызов функции
invokeRes: me.test_ExtLibCalc( _str, _a1, _x1, _a2, _x2, &_y1, &_y2, &_err, _
length);
```

3. Возможности языка SePlatform.Om в сигнальной модели

Все стандартные возможности языка SePlatform.Om применимы к сигнальной модели. Помимо стандартных возможностей языка, существуют некоторые надстройки и функции, которые работают исключительно в сигнальной модели данных.

3.1. Относительная адресация в сигнальной модели

Относительная адресация применяется при необходимости обратиться к далекому уровню дерева сигналов. Для работы с объектами сигнальной модели используются следующие ключевые слова и символы:

- me - указатель на текущий сигнал;
- @N - обращение к N-му уровню дерева выше текущего сигнала:
  - @0 - аналогично me, т.е. текущий сигнал
  - @1 - родительский узел дерева
  - @2 - родитель родителя

ПРИМЕР

Пусть в дереве существует сигнал. Его полный тег: AK.DMN.LU1.NPS1.TANK1.SW1.State. Тогда:

- me - сам сигнал: AK.DMN.LU1.NPS1.TANK1.SW1.State
- @1 - родительский узел дерева: AK.DMN.LU1.NPS1.TANK1.SW1
- @2 - "прыжок" на два уровня вверх (родитель родителя): AK.DMN.LU1.NPS1.TANK1
- @3 - "прыжок" на три уровня вверх: AK.DMN.LU1.NPS1.

$ - символ «доллар» применяется в ситуациях, когда тег сигнала противоречит стандартным правилам написания (наличие пробелов или запрещенных символов). Символ "$" указывает компилятору, что написанная после него строка (заключенная в двойные кавычки) является путем до сигнала в иерархии дерева.

```
//В теге сигнала содержатся пробелы, нужно использовать $
X: uint1 = $"Уровень 1.Уровень 2.Уровень 3.Статус задвижки";
```

3.2. Работа с сигнальными свойствами

При написании скриптов для сигнальной модели можно обращаться к свойствам сигналов:

- по имени:

```
AKM.Signal.Description
```

- по номеру:

```
AKM.Signal.$"101"
```

ОБРАТИТЕ ВНИМАНИЕ

- Свойство должно существовать
- Обращение по имени доступно только для именованных свойств
- Для записи доступны только свойства из диапазона 5100-5108

ПРИМЕР

Использование свойства Description (101) в качестве аргумента функции:

```
Str.SubString(sig.$"101", 2)
```

3.3. Влияние различных операций на сопутствующую информацию

Каждый сигнал дерева в модуле вычислений представлен объектом с сопутствующей информацией:

- Value - значение
- Quality - качество
- Timestamp - метка времени

При выполнении любой операции (арифметической, логической, операции сравнения и т.д), где в качестве операндов выступают сигналы, вычисляется как значение (результат операции), так и качество и метка времени.

- Правила вычисления качества значения для всех операций одинаковы. Результат выполнения любой операции может иметь одно из трех значений:
  - **«200»** (OPC_QUALITY_CALCULATED) - если качество всех аргументов достоверное (>=192) и не возникло переполнений;
  - **«88»** (OPC_QUALITY_SUBNORMAL) - если качество хотя бы одного из аргументов недостоверное (<192) и не возникло переполнений;
  - **«84»** (OPC_QUALITY_EGU_EXCEEDED) - если в результате операции зафиксировано переполнение
- Правила вычисления метки времени:
  - Если модуль работает в режиме Now, то результату операции будет присвоена текущая метка времени сервера.
  - Метка времени события, которое привело к срабатыванию триггера. Определяется по максимальной метке времени операндов операции.

3.4. VQT-функции

VQT-функции изменяют значение (Value), качество (Quality) и метку времени (Timestamp):

- VQ

Изменяет значение и качество сигнала, метка времени не изменяется

```
<Путь до сигнала> = VQ(Value, Quality)
```

- VT

Изменяет значение и метку времени сигнала, качество не изменяется

```
<Путь до сигнала> = VT(Value, Timestamp)
```

- VQT

Изменяет значение, качество и метку времени сигнала.

```
<Путь до сигнала> = VQT(Value, Quality, Timestamp)
```

Параметры:

Параметр	Тип	Описание
Value	Тип параметра должен совпадать с типом сигнала или неявно к нему приводиться	Новое значение сигнала
Quality	uint4	Новое качество сигнала
Timestamp	uint8	Новая метка времени сигнала

ПРИМЕР

Установить значение и качество

```
Level1.Level2.Signal = VQ(65536,192);
```

Установить значение и качество логического сигнала

```
Level1.Level2.Enabled = VQ(true, 192);
```

Установить значение и метку времени

```
Level1.Level2.Signal = VT(50, 130004837290000000);
```

Установить значение и метку времени строкового сигнала

```
Level1.Level2.Str = VT("Message", 130004837290000000);
```

Установить сигналу значение, метку времени и качество

```
Level1.Level2.Signal = VQT(90,145,131207739510000000);
```

3.5. Оператор commit

Оператор commit предназначен для форсированной передачи в ядро сервера всех значений сигналов, измененных в скрипте до вызова оператора.

Синтаксис оператора в общем виде:

```
<Блок обработки сигналов>
commit X=Y
```

Где X=Y - финальная операция присваивания и фиксации в ядро, выполняемая в последнюю очередь.

ПРИМЕР

Без оператора commit:

```
AK.DMN.Counter1 = 0; //присваиваем значение сигналу
while(AK.DMN.Counter1<10)
{
  AK.DMN.Counter1 += 1;  //увеличиваем значение сигнала
}
//Цикл сделает 10 итераций
//Но в сигнал AK.DMN.Counter1 запишется только финальное значение = 10
```

ПРИМЕР

С оператором commit:

```
AK.DMN.Counter2 = 0; //присваиваем значение сигналу
while(AK.DMN.Counter2<10)
{
  commit AK.DMN.Counter2 += 1; //увеличиваем значение сигнала и фиксируем в ядро
}
//За время работы цикла значение сигнала будет изменено 10 раз (от 1 до 10)
```

4. API предоставления данных

Модуль предоставляет различную информацию о конфигурации сервера:

- идентификаторы сигналов
- полные и короткие имена сигналов
- значения свойств сигналов
- идентификаторы родительских объектов, а также все ссылки на узел

Данный функционал реализован в виде API сервера, включающий набор функций. Вызов функций осуществляется посредством запросов в формате JSON. Возможность вызова функций предоставляет модуль OPC DA Server при помощи OPC DA-клиента, подключенного к серверу.

4.1. Набор функций API сервера

API сервера включает следующий набор функций для получения информации о конфигурации сервера:

Название функции	Параметры запроса	Назначение функции
Сигнальная модель		
GetIdByTagName (uint4)	tagname (string)	Получение ID по имени сигнала
GetTagNameById (string)	nodeid (uint4)	Получение полного имени сигнала по ID
GetShortNameById (string)	nodeid (uint4)	Получение короткого имени сигнала по ID
ReadProperty (variant)	nodeid (uint4) propid (uint4)	Чтение значения свойства сигнала по ID свойства
ReadProperty (variant)	nodeid (uint4) propname (string)	Чтение значения свойства сигнала по имени свойства
Объектная модель		
GetParentObjectId (uint4)	nodeid (uint4)	Получение ID родительского объекта по ID объекта
FindAllReferences (uint4)	nodeid (uint4)	Найти все ссылки (ID) на узел по ID узла

4.2. Формат запросов JSON

Формат запроса JSON имеет вид:

```
{
  "transaction":"Идентификатор_вызова",
  "request":{
    "target":"Имя_узла",
    "method":"Функция_узла",
    "input":{
      "Имя_параметра":"Значение_параметра"
```

```
    }
  }
}
```

Значение	Описание
Идентификатор_вызова	Пользовательский идентификатор, задаётся пользователем и может иметь любое значение
Имя_узла	Узел, представляющий API сервера
Функция_узла	Имя вызываемой функции API сервера
Имя_параметра	Параметр, по которому производится запрос (параметров может быть несколько)
Значение_параметра	Значение параметра, по которому производится запрос. Может быть задано в виде массива, если необходимо получить информацию по нескольким параметрам

Формат ответа на запрос JSON имеет вид:

```
{
  "transaction":"Идентификатор_вызова",
  "result":{
    "return":"Результат",
    "output":"Выходные параметры"
  }
}
```

Значение	Описание
Идентификатор_вызова	Пользовательский идентификатор, заданный пользователем при запросе
Результат	Возвращаемые значения вызванной функцией
Выходные параметры	Выходные параметры каждого вызова функции

ПРИМЕР

Запрос:

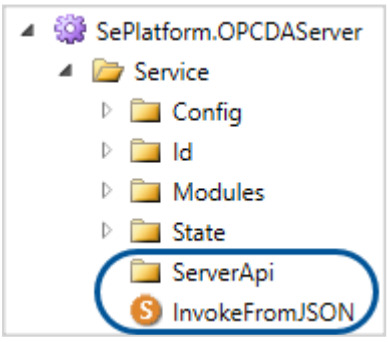
```
{
  "transaction":"19Cnd23",
  "request":{
    "target":"Service.ServerApi",
    "method":"GetIdByTagName",
    "input":{
      "tagname":"NPS.MNS1.PT001_1.Value"
    }
  }
}
```

Ответ:

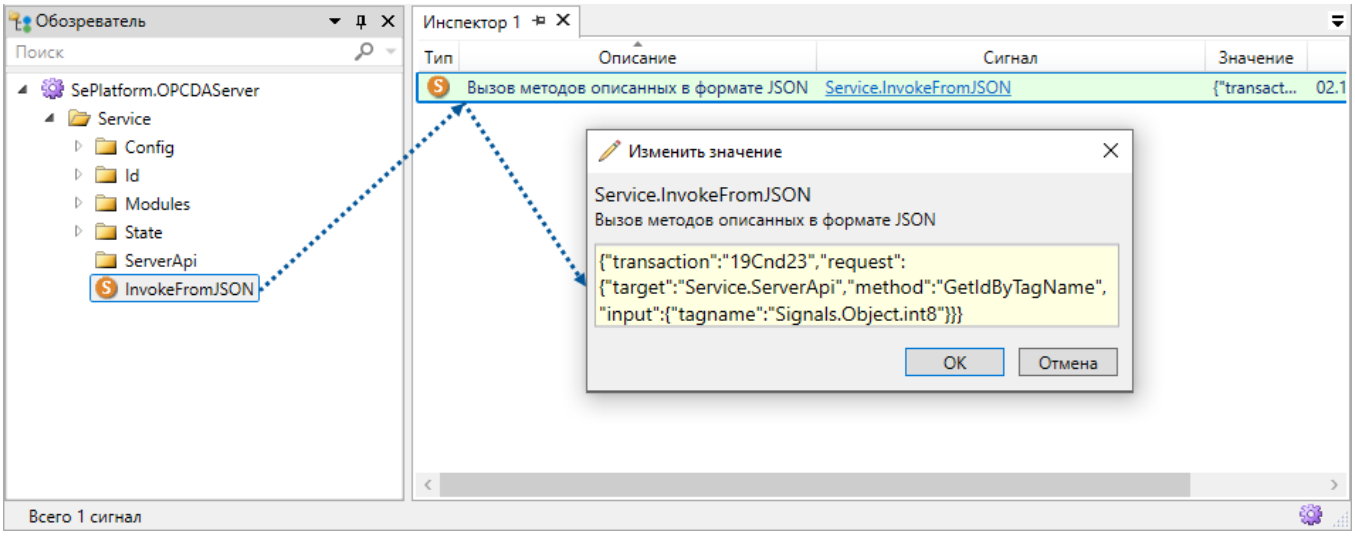
```
{
  "transaction": "19Cnd23",
  "result":{
    "return":457
  }
}
```

4.3. Вызов функций API сервера

Вызов функций API сервера осуществляется модулем логики, а модуль OPC DA Server предоставляет возможность вызова функций посредством подключенного OPC DA-клиента. Модуль логики динамически создаёт узел Service.ServerApi, представляющий API сервера, а модуль OPC DA Server создаёт динамический сигнал Service.InvokeFromJSON типа string.



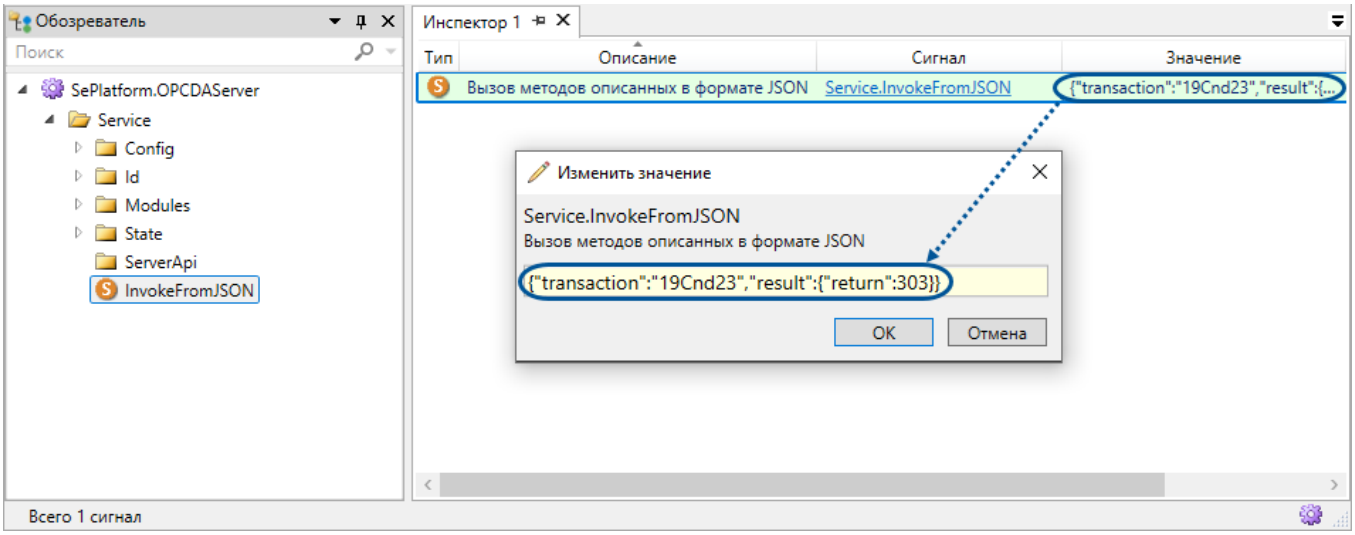
Для получения необходимой информации о конфигурации сервера необходимо подключиться к серверу с помощью OPC DA клиента и в значение сигнала Service.InvokeFromJSON записать запрос в формате JSON, который вызывает соответствующую функцию Service.ServerApi.



Запрос в формате JSON прописывается в значение сигнала в одну строку, например:

```
{"transaction":"19Cnd23","request":{"target":"Service.ServerApi","method":"GetIdByTagName",
"input":{"tagname":"Signals.Object.int8"}}}
```

При записи запроса в сигнал Service.InvokeFromJSON происходит вызов соответствующей функции, после выполнения которой ответ в формате JSON записывается в тот же сигнал.



4.4. Примеры запросов

Сигнальная модель:

- Запрос на получение значения идентификатора сигнала NPS.MNS1.PT001_1.Value:

```
{"transaction":"json_1","request":
{"target":"Service.ServerApi","method":"GetIdByTagName","input":
{"tagname":"NPS.MNS1.PT001_1.Value"}}}
```

- Ответ на запрос содержит значение идентификатор сигнала 457:

```
{"transaction":"json_1","result":{"return":457}}
```

- Запрос на получение полного имени сигнала по идентификатору 457:

```
{"transaction":"json_2","request":
{"target":"Service.ServerApi","method":"GetTagNameById","input":{"nodeid":457}}}
```

- Ответ на запрос содержит полное имя сигнала NPS.MNS1.PT001_1.Value:

```
{"transaction":"json_2","result":{"return":"NPS.MNS1.PT001_1.Value"}}
```

- Запрос на получение короткого имени сигнала по идентификатору 457:

```
{"transaction":"json_3","request":
{"target":"Service.ServerApi","method":"GetShortNameById","input":{"nodeid":457}}}
```

- Ответ на запрос содержит короткое имя сигнала Value:

```
{"transaction":"json_3","result":{"return":"Value"}}
```

- Запрос на получение описания сигнала с идентификатором 457 по идентификатору свойства (101 (Description)):

```
{"transaction":"json_4","request":
{"target":"Service.ServerApi","method":"ReadProperty","input":
{"nodeid":457,"propid":101}}}
```

- Ответ на запрос содержит описание сигнала:

```
{"transaction":"json_4","result":{"return":"Давление ДТ на приёме МНС (т.1). Текущее
значение"}}
```

- Запрос на получение массива информации о сигнале с идентификатором 457 по идентификаторам свойств: ведение истории (9001 (Historizing)), описание (101 (Description)), единицы измерения (100 (EUnit)):

```
{"transaction":"json_5","request":
{"target":"Service.ServerApi","method":"ReadProperty","input":{"nodeid":457,"propname":
[9001,101,100]}}}
```

- Ответ на запрос содержит значения всех запрошенных свойств:

```
{"transaction":"json_5","result":{"return":[true,"Давление ДТ на приёме МНС (т.1).
Текущее значение","МПа"]}}
```

- Запрос на получение описания сигнала с идентификатором 457 по названию свойства (101 (Description)):

```
{"transaction":"json_6","request":
{"target":"Service.ServerApi","method":"ReadProperty","input":
{"nodeid":457,"propname":"Description"}}}
```

- Ответ на запрос содержит описание сигнала:

```
{"transaction":"json_6","result":{"return":"Давление ДТ на приёме МНС (т.1). Текущее
значение"}}
```

- Запрос на получение массива информации о сигнале с идентификатором 457 по именам свойств: ведение истории (9001 (Historizing)), описание (101 (Description)), единицы измерения (100 (EUnit)):

```
{"transaction":"json_7","request":
{"target":"Service.ServerApi","method":"ReadProperty","input":{"nodeid":457,"propid":
[Historizing,Description,EUnit]}}}
```

- Ответ на запрос содержит значения всех запрошенных свойств:

```
{"transaction":"json_7","result":{"return":[true,"Давление ДТ на приёме МНС (т.1).
Текущее значение","МПа"]}}
```

Объектная модель:

- Запрос на получение идентификатора родительского объекта для дочернего объекта с идентификатором 103:

```
{"transaction":"json_8","request":
{"target":"Service.ServerApi","method":"GetParentObjectId","input":{"nodeid":103}}}
```

- Ответ на запрос содержит идентификатор родительского объекта:

```
{"transaction":"json_8","result":{"return":101}}
```

- Запрос на получение идентификаторов всех ссылок на узел с идентификатором 103:

```
{"transaction":"json_9","request":
{"target":"Service.ServerApi","method":"FindAllReferences","input":{"nodeid":103}}}
```

- Ответ на запрос содержит идентификаторы ссылок на узел:

```
{"transaction":"json_9","result":{"return":"[57,115,124]"}}
```

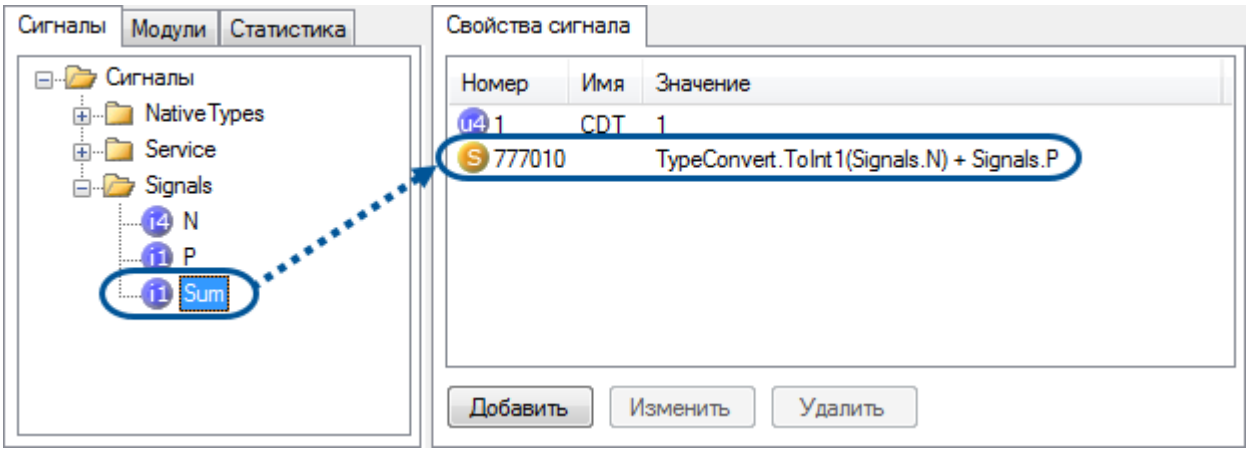
5. Пример использования модуля

5.1. Вычисление с помощью формулы

Имеем сигнал Signals.N типа Int4 и сигнал Signals.P типа Int1. Необходимо вычислять сумму их значений и результат сложения записывать в сигнал Signals.Sum типа Int1.

1. Поскольку формула не может содержать двух значений с разными типами следует преобразовать тип значения сигнала Signals.N к типу Int1. Для этого следует воспользоваться функцией TypeConvert.ToInt1 (Signals.N).
2. Для сигнала Signals.Sum добавить свойство 777010 и записать в него значение

```
TypeConvert.ToInt1 (Signals.N) + Signals.P.
```



3. Для проверки корректности задания формулы подписаться на сигналы Signals.N, Signals.P, Signals.Sum с помощью любого OPC клиента и изменить значение сигнала Signals.N или Signals.P.

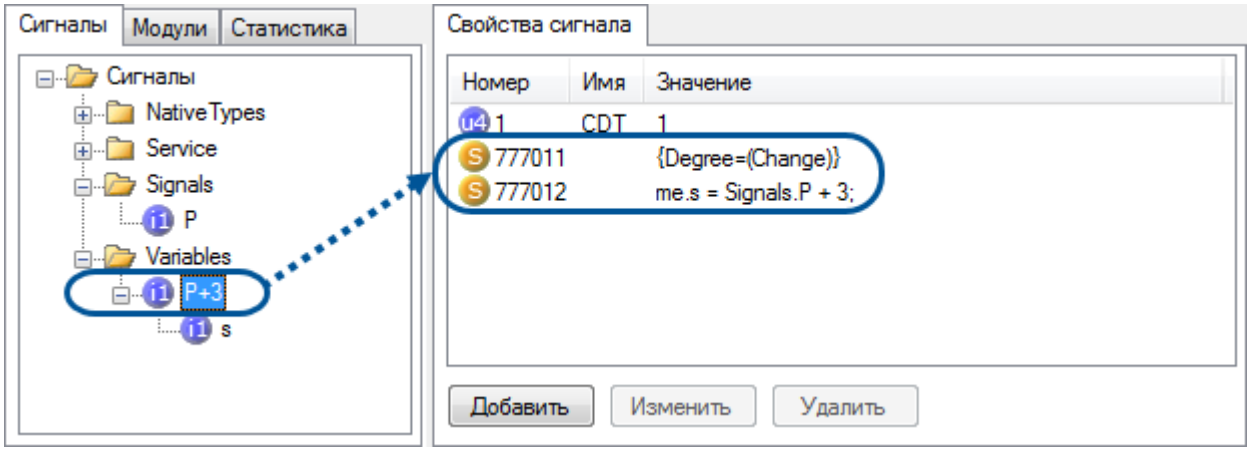
5.2. Вычисление с помощью процедуры

Имеем сигнал Variables.P+3, при изменении значения которого необходимо увеличивать значение сигнала Signals.P на 3 и записывать результат в сигнал Variables.P+3.s.

1. Поскольку основным условием расчета является изменение значения сигнала Variables.P+3, то этому сигналу следует добавить свойство 777011 со значением {Degree=(Change)} и свойство 777012, в которое будет записана процедура.
2. Присвоить значению сигнала Variables.P+3.s выражение Signals.P + 3. Полный тег сигнала Variables.P+3 можно заменить на ключевое слово me.

3. Итоговая запись в свойство 777012 будет иметь вид:

```
C2: var = 3 + Signals.P;
me.s = C2;
```



4. Для проверки корректности задания процедуры подписаться на сигналы Signals.P, Variables.P+3, Variables.P+3.s с помощью любого OPC клиента и изменить сначала значение сигнала Signals.P, а затем Variables.P+3. В результате значение сигнала Variables.P+3.s будет пересчитываться.

Список терминов и сокращений

Процедура	Набор последовательных действий (операций), выполняемых при наступлении определенных условий.
Триггер	Условие, наступление которого должно приводить к выполнению процедуры.
Формула	Частный случай процедуры, представляющей из себя выражение, определяющее значение сигнала или свойства, для которого она задана.